University of Pisa
Computer Science

Machine Learning

Based on Prof. Alessio Micheli's lectures

September 17, 2021

Alessandro Cudazzo

Giulia Volpi

ACADEMIC YEAR 2019/2020

Notes

These notes are intended only as support for the study of the subject and as slides side notes. They do not cover the whole program but only a part. In order to compensate for the missing parts, we recommend the use of slides and books provided by Prof. Micheli for each topic.

We hope these notes will help future students and if you find any mistakes or want to help extend these notes, feel free to do so in the GitHub repo.

What you will find here: Introduction to ML, Linear Model and K-NN, Neural Networks, Validation, Statistical Learning Theory (STL).

What's missing: Concept Learning, SVM, Bias Variance, Deep Learning (CNN, Deep, Rand), SOM, Bayes Learning, RNN, SDL.

Contents

1 Introduction

> The problem of learning is arguably at the very core of the problem of intelligence, both biological and artificial
>
> *Poggio, Shelton, AI Magazine 1999*

1.1 What is ML

The field of **machine learning** is concerned with the question of how to construct computer programs that automatically improve with experience.
In recent years many successful machine learning applications have been developed, ranging from data-mining programs that learn to detect fraudulent credit card transactions, to information-filtering systems that learn users' reading preferences, to autonomous vehicles that learn to drive on public highways.
At the same time, there have been important advances in the theory and algorithms that form the foundations of this field. Machine learning draws on concepts and results from many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory.

Machine Learning has emerged as an area of research combining the aims of *creating computers that could learn* (AI - Build adaptive/personalized intelligent systems) and new powerful *adaptive/statistical tools* with rigorous foundation in computational science for data analysis.
Learning is the major challenge and a strategic way to provide intelligence into the systems.

Definition 1.1 (Learning Computer)
*A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

The ML studies and proposes methods to build (infer) a model (dependencies / functions / hypotheses) from examples of observed data

- That fits the known examples.
- Able to generalize with reasonable accuracy for new data (according to verifiable results, under statistical and computational conditions and criteria).
- Considering the expressiveness and algorithmic complexity of the models and learning algorithms.

Example 1.1 (Inferring general functions from know data)
Two famous examples of ML tasks:

- *Handwriting Recognition: x (Data from pen motion) and $f(x)$ (Letter of the alphabet)*
- *Face recognition: x (Bitmap picture of person's face) $f(x)$ (Name of the person)*

So we have some opportunity (if useful) and awareness (needs and limits):

Utility of predictive models: (in the following cases):

- no (or poor) theory (or knowledge to explain the phenomenon)
- uncertain, noisy or incomplete data (which hinder formalization of solutions)

Requests:

- source of training experience (representative data)
- tolerance on the precision of results

1.2 Overview: Components of a ML System

Main components of a ML system, Framework as a guide to the key design choices:

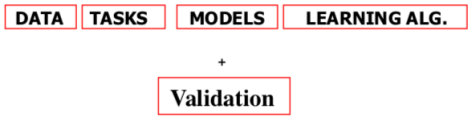


Figure 1.1: Components of a ML system

1.3 Data

The data represent the available facts (experience). Representation problem: to capture the structure of the analyzed objects. **Type**: Flat, Structured, ...

We will generally use flat data:
E.g. **Flat DATA** (attribute-value language): fixed-size vectors of properties (features), single table of tuple (measurements of the objects), Attributes can be discrete or continuous

Fruits	Weight	Cost $	Color	Bio
Fruit 1 (lemon)	2.1	0.5	y	1
Fruit 2 (apple)	3.5	0.6	r	?

→ Attributes (discrete/continuous)

- Categorical/continuous, missing data,...
- Preprocessing: e.g. Variable scaling, encoding*, selection...

Of course Data type can be different, usually there is a Data Preprocessing step, e.g. Variable scaling, encoding, selection... For this step, see in Data Mining: *Data Understanding* and *Data Preparation*.

Example and terminologies
Let's see a example with some Medical records in the flat case:

Patients	Age	Smoke	Sex	Lab Test
Pat 1	101	0.8	M	1
Pat 2	30	0.0	F	?

→ Attributes (discrete/continuous)

- Each row ($\mathbf{x}$, vector): example, pattern, instance, sample,...
- Dimension of data set: number of examples l
- Dimension (of $\mathbf{x}$): number of features n
- If we will index the features/inputs/variables by j : variable x_j is (typically) the j-th feature/property/attribute/element of $\mathbf{x}$.

but may be to simplify we need to use subscript index for other meanings

- $\mathbf{x}_p$ is (typically) the p-th pattern/example/raw ($\mathbf{x}$ bold is a vector)
- x_{pj} for example can be the attribute j of the pattern p

DATA Encoding
For the **Flat case** we can have **Numerical encoding for categories** e.g.:

- 0/1 (or –1/+1) for 2 classes
- 1,2,3... Warning: grade of similarity (1 vs 2 or 3): useful for "order categorical" variables (e.g small, medium, large)

- 1-of-k (or 1-hot) encoding: useful for symbols

A	1	0	0
B	0	1	0
C	0	0	1

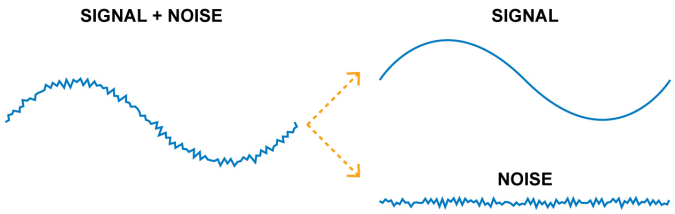
Useful both for input or output variables

This is useful because the scalar product between two vectors representing two coded symbols is zero. The vectors are orthogonal, this implies that there is no relation between the category symbols.

Another types of Data:
E.g. **Structures DATA**: Sequences (lists), trees, graphs, Multi-relational data (table) (in DB)
Examples: images, microarray, temporal data, strings of a language, DNA e proteins, hierarchical relationships, molecules, hyperlink connectivity in web pages, ...

Further terminologies

- **Noise**: addition of external factors to the stream of target information (signal); due to randomness in the measurements, not due to the underlying law: e.g. Gaussian noise



- **Outliers**: are unusual data values that are not consistent with most observations (e.g. due to abnormal measurements errors). Some operation can be perform: outlier detection and preprocessing (removal) or we have to use some Robust Modeling Methods.
- **Feature selection**: selection of a small number of informative features: it can provide an optimal representation for a learning problem

1.4 Task

The task defines the purpose of the application: Knowledge that we want to achieve? Which is the helpful nature of the result? What information are available?
There are two main categories of tasks:

- **Predictive** (Classification, Regression): function approximation (build a function from examples)

- **Descriptive** (Cluster Analysis, Association Rule): find subsets or groups of unclassified data

We can usually divide tasks in *Supervised Learning* and *Unsupervised Learning*:

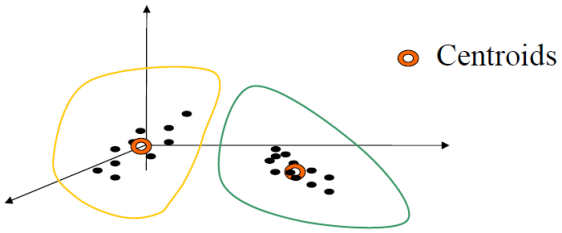


Figure 1.2: An Example of Clustering, Partition of data into clusters (subsets of "similar" data)

Supervised Learning

Given a training example as $< input, output >=< \mathbf{x}, d >$ (**labeled example**) for an unknown function f, the aim is to find a *good* approximation to f (an **hypothesis** h that can used for prediction on unseen data **x'**).
Target d (or t or y) is given by the teacher according to a $f(\mathbf{x})$ (unknown function), d is usually a numerical/categorical label:

- *Classification*: discrete value outputs:

$$f(\mathbf{x}) \in \{1, 2, \ldots, K\} \text{ classes (discrete-valued function).}$$

- *Regression*: real continuous output values (approximate a real-valued target function)

Note: we can use the same model, we only have to change the domain of the output (Unified vision thanks to the formalism of func. approximation).

Unsupervised Learning

Given a training set of **unlabeled data** $< \mathbf{x} >$, the aim is to find a *natural groupings* in the set. Some taks are:

- Clustering, see figure 1.2
- Dimensionality reduction/ Visualization/Preprocessing
- Modeling the data density

1.4.1 Classification

(Supervised) Classification: Patterns (features vectors) are seen as members of a class and the goal is to assign the patterns observed classes (label).
So, the aim is to find a *good* approximation to $f(\mathbf{x})$ (an **hypothesis** h) that can be used to return a prediction on unseen data **x'** and tell what class **x'** belongs to, see figure 1.3.

If the number of classes is 2, the approximation of $f(\mathbf{x})$ is a boolean function (binary classification, **concept learning**), instead if the number of classes is greater than 2, we talk about *multi-class problem* $(C_1, C_2, \ldots, C_k)$.

The classification may be viewed as the allocation of the input space in decision regions (e.g. $0/1$) and we want to find a linear separator on an instance space $\mathbf{x} = (x_1, x_2)$in $\mathbb{R}^2$ where $f(x) = 0/1$(or $-1/+1$).

In figure 1.4a we can see an hyperplane which separates the points and this is a hypothesis $h(x)$ (an approximation to $f(\mathbf{x})$) found in the space of hypotheses H (set of dichotomies induced by hyperplanes), see figure 1.5.

From DATA to TASK (e.g. classification)

Patients	Age	Smoke	Sex	Lab Test
Pat 1	101	0.8	M	1
Pat 2	30	0.0	F	?

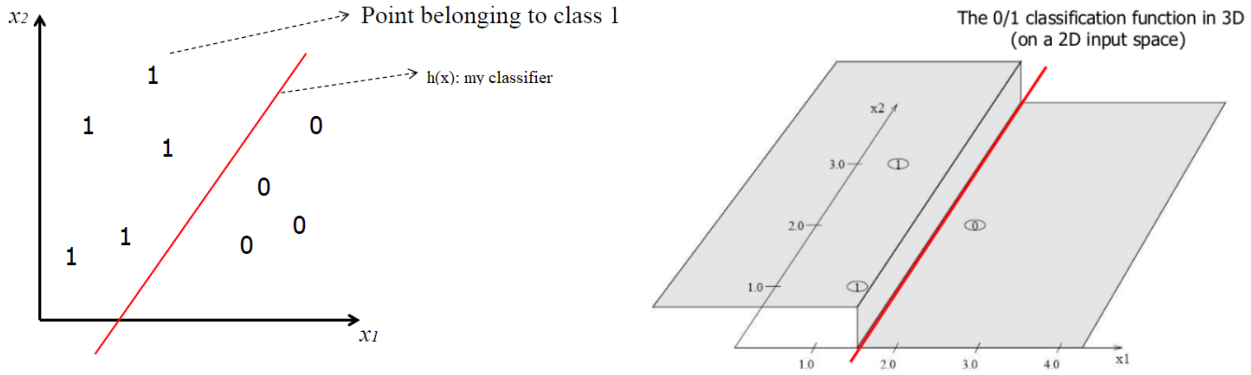
Target: diagnose
+
-

f

$\boldsymbol{x}$: Input space

Terminology in statistics:
- Inputs are the "independent variables"
- Outputs are the "dependent variables" or "responses"

Figure 1.3: A classification task: we want to get an approximation to $f(\mathbf{x})$, we want to understand if future patients are positive or negative to a certain diagnosis



(a) Separating (hyper)plane : x.

(b) Geometrical 3D view: classifier

Figure 1.4: An example of classification, that use a linear separator on the instance space.

$$\boldsymbol{wx} + w_0 = w_1 x_1 + w_2 x_2 + w_0 = 0$$

$$h(\boldsymbol{x}) = \begin{cases} 1 & \_if \quad \boldsymbol{wx} + w_0 \geq 0 \\ 0 & \_\_\_\_\_otherwise \end{cases}$$

or

$$h(\boldsymbol{x}) = sign(\boldsymbol{wx} + w_0)$$

Linear threshold unit (LTU)

Indicator functions

Figure 1.5: Hyperplanes classifier

1.4.2 Regression

Process of estimating of a real-value function on the basis of finite set of noisy samples (supervised task), we have known pairs $(x, f(x) + \text{random noise})$.
Goal Task: find a function that approximate the data in figure 1.6:

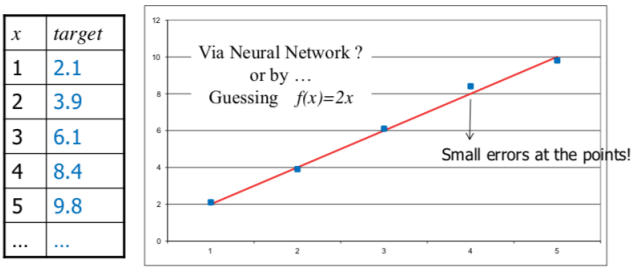
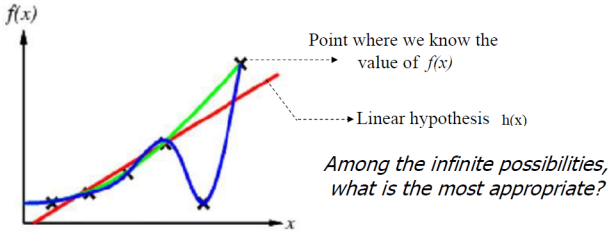


Figure 1.6: A example of regression

Regression: x = variables (e.g real values), $(f(x) + \text{random noise})$ real values that we know, so we want to find a curve that best fit the data: *curve fitting*. An example of curve fitting is a **linear hypothesis**:



$h_w(x) = w_1 x + w_0 = 0.2x - 0.4$ but we have to choose the best one with some criteria.

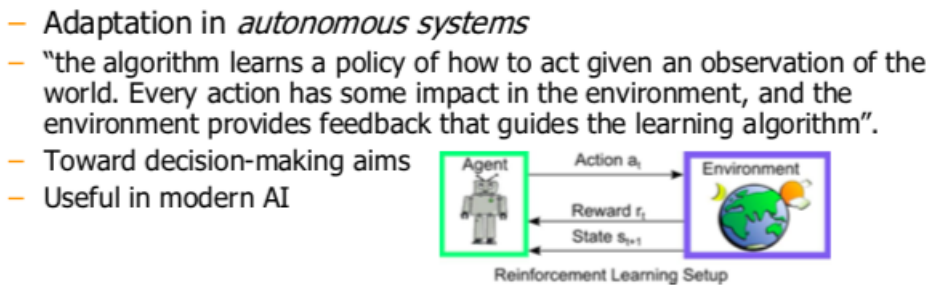
1.4.3 Other Tasks

There are other types that we will not discuss, like:

- **Dimensionality reduction** (in unsupervised Learning)

$$< x_1, x_2, \ldots, x_n > \rightarrow < x_1, x_2, \ldots, x_m > \text{ with } m < n$$

- **Reinforcement Learning**(learning with right/wrong critic): This is usually a supervised learning task with some police!



- **Semi-supervised learning**: combines both labeled and unlabeled examples to generate an appropriate function or classifier.

1.5 Models

The aim is to capture/describes the relationships among the data (on the basis of the task) by a "language". The "language" is related to the representation used to get knowledge. It defines the class of functions that the learning machine can implement (*hypothesis space*).
E.g. set of functions $h(x, w)$, where w is the (abstract) parameter.

Definition 1.2 (Model main terms)
The terminologies associated to the model is:

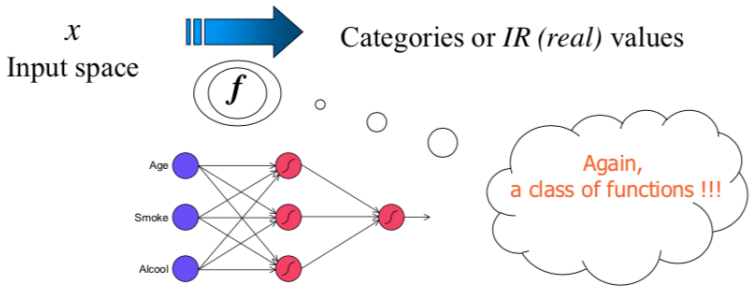
- ***Training examples*** *(supervised learning):*
  An example of the form $(x, f(x))$. x is usually a vector of features, $t = f(x)$ is called the target value.
- ***Target function****: The true function f (that we don't have! we want to approximate it!)*
- ***Hypothesis****: A proposed function h believed to be similar to f. An expression in a given* language *that describes the relationships among data.*
- ***Hypotheses space (H)****: The space of all hypotheses that can, in principle be output by the learning algorithm (set of functions $h(x, w)$, where w is the parameter), this is a Hilbert Space.*

Some models are shown here:

Just to have a preview of different *representation* of hypothesis
(because you already know the language of equations, logic, probability):

- **Linear models** (representation of H defines a continuously parameterized space of potential hypothesis);
  each assignment of w is a different hypothesis, e.g:
  - $h(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + w_0)$ — *binary classifier*
  - $h_w(x)=w_1x+w_0$ $\quad E.g.\ h_w(x)= 2x+150$ — *simple linear regression*
- **Symbolic Rules**: (hypothesis space is based on discrete representations); different rules are possible , e.g:
  - *if $(x_1=0)$ and $(x_2=1)$ then $h(\mathbf{x})=1$* — *binary classifier*
  - *else $h(\mathbf{x})=0$*
- **Probabilistic models**: estimate $p(x,y)$
- K Nearest neighbor regression: Predict mean y value of nearest neighbors (memory-based)

Another example: we will see a **neural networks**, beyond the neurobiological inspiration, as a computational model for the treatment of data, capable of approximating complex (non-linear) relationships between inputs and outputs.



1.5.1 Pardigms and methods (Languages for H)

- Symbolics and Rule-based (or discrete H)
  - Conjuction of literals, Decision trees (propositional rules)
  - Inductive grammars, Evolutionary algorithms, ...
  - Inductive Logic Programming (first order logic rules)
- Sub-symbolic (or continuous H)
  - Linear discriminant analysis, Multiple Linear Regression, LTU
  - Neural networks
  - Kernel methods (SVMs, gaussian kernels, spectral kernels, etc)
- Probabilistic/Generative
  - Traditional parametric models (density estimation,discriminant analysis,polynomial regression,...)
  - Graphical models: Bayesian networks,naive Bayes, PLSA, Markov models, Hidden Markov models, ...
- Instance-based
  - Nearest neighbor

1.5.2 How many models?

Theorem 1.1 (No Free Lunch Theorem)
There is no universal "best" learning method (without any knowledge, for any problems,...): if an algorithm achieves superior results on some problems, it must pay with inferiority on other problems. In this sense there is no free lunch.

NOTE: However, not all the models are equivalent: (instead of assuming a specific form for the target function (parametric models in classical Statistics)).
The course provide:

- A set of models.
- The critical instrument to compare them.

The core of ML is on flexible approaches that can in principle approximate arbitrary functions (universal approximation property).

NOTE: Typical powerful models of ML are Neural Networks, always remeber that "power is nothing without control": we need of inductive principia for the control of the complexity.

1.6 Learning Algorithms

A Learning Algorithms is basing on data, task and model!
(Heuristic) search through the hypothesis space H ($h \in H$) of the **best hypothesis** (Typically searching for the h with the minimum "error", the best approximation to the unknown target function). Practically we want to find parameters w that give the best h.

Example 1.2
Learning could mean to search the best $\boldsymbol{w}$ for the linear models or best rules for symbolic model.

H may not coincide with the set of all possible functions and the search can not "be exhaustive": it needs to make assumptions (we will see the role of *Inductive bias*), see figure 1.7.

Learning (terminologies): According to the different paradigms/contexts "learning" can be differently termed or have different acceptations:

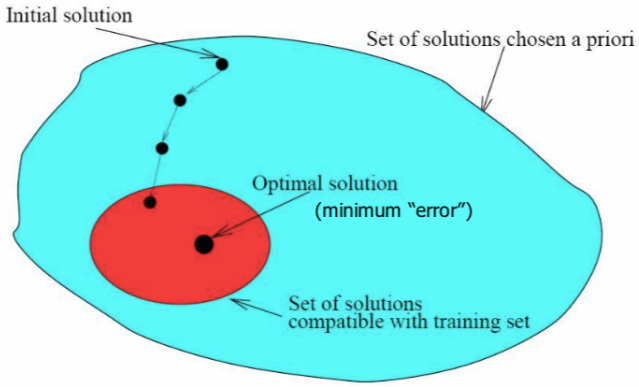


Figure 1.7: (Local) search on the hypothesis space: each point is a function, we set an initial condition and we move to a set of optimal solution (red).

- Inference (statistics)
- Inference: Abduction/Induction (logic)
- Adapting (biology, systems)
- Optimizing (mathematics)
- Training (e.g. Neural Networks)
- Function approximations (mathematics)

Can be more specifically found in other sub-fields:

- Regression analysis (statistics), curve fitting (math, CS), ...
- Or using other terminologies e.g. "Fitting a multivariate function"

1.7 Role of inductive bias

In order to set up a model and a learning algorithm we can make assumptions (about the nature of the target function) concerning either

- Constraints in the model (in the hypothesis space H, due to the set of hypotheses that we can express or consider) (Language Bias). For example we can search only on linear function.
- Constraints or preferences in learning algorithm/search strategy (Search Bias). For example we could choose to search only some $\boldsymbol{w}$ for the linear search.
- Or Both.

We will see that such assumptions are strictly need to obtain an useful model for the ML aims, i.e. a model with generalization capabilities.
We start to discuss it within examples in discrete hypotheses spaces (rules) , learning a concept (a Boolean function) [Mitchell chapt. 2]

Example 1.3
x is a "cat" if h cat (x) =1, otherwise is 0 for x in "animals"

Example 1.4
Learning a boolean function (boolean input/output) given the True table . . . slides 119 and 120

ADD SLIDES 119—¿ 128

1.8 Task + Model = Loss

We have to find a "**good**" approximation to f from examples. How to measure the quality of the approximation?

We want to measure the "distance" between $h(x)$ (output of the model for input $\mathbf{x}$) and d.

Minimization of errors in training, check of errors in test

We will use:

- *Loss function*: $L(h(\mathbf{x}), d)$ (high value means poor approximation)
- The Error (or Risk) is an expected value of the loss:

$$\textit{Empirical Risk}: R_{emp} = \frac{1}{l}\sum_{i=1}^{l} L(h(\mathbf{x}_i), d_i).$$

  e.g. a "sum" or mean of the loss over the set of samples.

We will change the loss function L for different tasks.

Common Tasks review

A possible classifications of common learning tasks specifying the (changing of the) nature of the loss function, output and hypothesis space.

- Survey of common learning tasks
- Nature of models (hypothesis spaces) for different class of tasks.

1.8.1 Loss Function example: Regression

In a regression task we want to predict a numerical value. We have an **Hypothesis Space** H that is a set of real-valued functions and the **Loss function** will measures the approximation accuracy/error between $h(x_i)$ and $d_i = f(x_i) + e$ (real value function + random error).

A common loss function for Regression (predicting a numerical value) is the *squared error*:

$$L(h(\mathbf{x}_i), d_i) = (d_i - h(\mathbf{x}_i))^2.$$

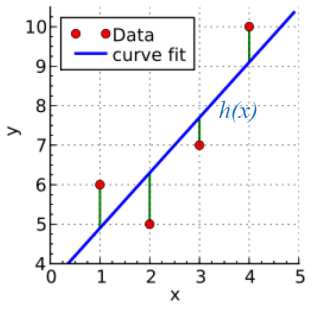
We use that error because we want the distance and a negative or positive value makes no different).

R_{emp}: the mean over the data set provide the *Mean Square Error* (MSE)

In the example we have $h(x)=w_1x+w_0$ as the blue line and in green the errors at the data points (x_i, y_i) (in red), where the target d_i for x_i is y_i in the example

The Mean Square Error (MSE) is the mean of the square of the green errors.

$E(\mathbf{w}) = \frac{1}{l}\sum_{p=1}^{l}(y_p - h_w(\boldsymbol{x}_p))^2$



Note: this plot is take from internet, I used different colors before: blue for the points (data) and red for the line in my previous plot.
Also the *y* are the desidered (target d) values

1.8.2 Loss Function example: Classification

Classification of data into discrete classes, d_i can be e.g. 0/1 and $h(x_i)$ will predict 0/1. So our **Loss Function** will measures the classification error:

$$L(h(\mathbf{x}_i), d_i) = \begin{cases} 0 & if \;\; h(\mathbf{x}_i) = d_i \\ 1 & otherwise \end{cases}$$

R_{emp}: The mean over the data set provide the number/percentage of misclassified patterns

Example 1.5
20 out of 100 are misclassified → 20% errors, i.e. 80% of accuracy

1.8.3 Loss Function example: Clustering and Vector Quantization

Goal: optimal partitioning of unknown distribution in x-space into regions (clusters) approximated by a cluster center or prototype. e.g see figure 1.2

H: a set of vector quantizers $x \rightarrow c(x)$ (continuosspace → discrete space)

Loss Function: would be the squared error distortion:

$$L(h(\mathbf{x}_i)) = \langle (\mathbf{x}_i - h(\mathbf{x}_i)), (\mathbf{x}_i - h(\mathbf{x}_i)) \rangle \quad (\text{ inner product })$$

Proximity of the pattern to the centroid of its cluster

1.8.4 Loss Function example: Density estimation

Density estimation (generative, "parametric methods") from an assumed class of density

- **Output**: a density e.g. normal distribution with mean m and variance $sigma^2$: $p(x \mid m, sigma^2)$
- **H**: a set of densities (e.g. m and $sigma^2$ are the two unknown *parameters*)
- A *common* **loss function** for density estimation:

$$L(h(\mathbf{x}_i)) = -\ln(h(\mathbf{x}_i)) \longrightarrow \text{We'll see later}$$

- Related to "maximizing the (log) likelihood function". [not hear]
- E.g. $P(x_1, x_2, x_3, \ldots \mid m, sigma^2)$

1.9 GENERALIZATION

Learning: search for a good function in a function space from known data (typically minimizing an Error/Loss), but how we can say that this model is good for our task after the learning step?

Note: This is the answer for the question: "What's the meaninng of Machine Learning?"

A model is Good with respect to the **generalization error**: how accurately the model predicts over novel samples of data (Error/Loss measured over new data)

So *Generalization* is a crucial point of ML. We will see the difference between "Easy to use ML tools" and the "correct/good use of ML".

So we have two main phases:

- **Learning** phase **(training, fitting)**: build the model from know data – *training data* (and bias).
- **Predictive** phase **(test)**: apply to new example (we take the input **x** and we compute the response by the model): evaluation of the predictive hypothesis, i.e. of the **generalization capability**.

The Predictive phase correspond to the Deployment/Inference use of the ML built model.

⇒ Generalization is a crucial point of ML!!!

Evaluation of performances for ML systems = predictive accuracy
Estimated by the error computed on the (Hold out) Test Set

Accuracy/performance estimation is a critical aspect, we can do it by:

- theory to understand under what mathematical conditions a model is able to generalize (*Statistical Learning Theory* [Vapnik]). For example: how many data is needed to generalized? It makes sense to use few data to generalize?
- Empirical (training, test) and cross-validation techniques

NB: The performance on training data provide an overoptimistic evaluation, so never test accuracy on training set!

A very important phase is **Validation**!

Definition 1.3 (Inductive Learning Hypotesis)
Any hypothesis h found to approximate the target function (f) well over a sufficiently large set of training examples will also approximate the target function well over any other unobserved examples.

This is the fundamental assumption of inductive learning (Supervised learning) (e.g. in concept learning) and we will have much more to say about it.

Definition 1.4 (Overfitting)
A model is in overfitting if it outputs an hypothesis $h(\cdot) \in H$ having true error ϵ and empirical error E, but There is another $h'(\cdot) \in H$ having $E' > E$ and $\epsilon' < \epsilon$.

1.9.1 Complexity on case of study

Let's take an example with a **parametric model** for regression: **Polynomial Curve Fitting Example**, with just one variable and we assume to know the target function, see figure 1.8a and as error function we will use the Sum-of-Squares Error, see figure 1.8b.

- The set of function is assumed as polynomials with degree M
- The complexity of the hypothesis increase with the degree M
- l = number of examples

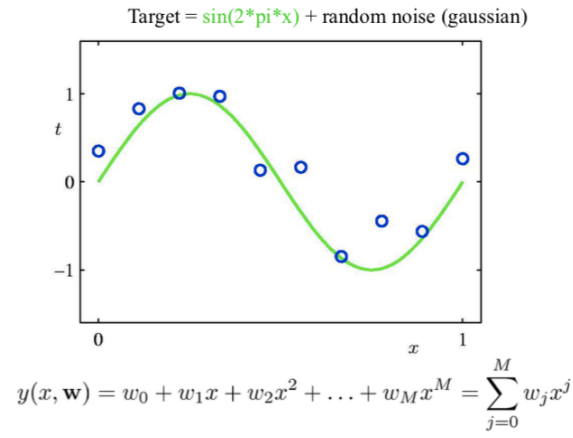
The Hypotesis Space in this case is:

$$h_w(x) = \sum_{j=0}^{M} w_j x^j.$$

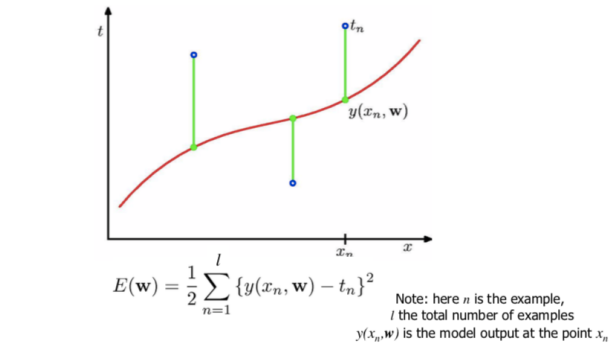
Warning: This is an artificial simplified task (unrealistic due to the use of just 1 input variable, the fact that we know the target function in advance, ...).

Let's analyze how good is a model if we increase the complexity and how change if the number of data, made available in the training phase, increases.
Case $l = 10$:
Assume that we have some few data (figure 1.8a) and let's see how our model will fit the data if we increase the complexity of the model.

Let's see some cases:



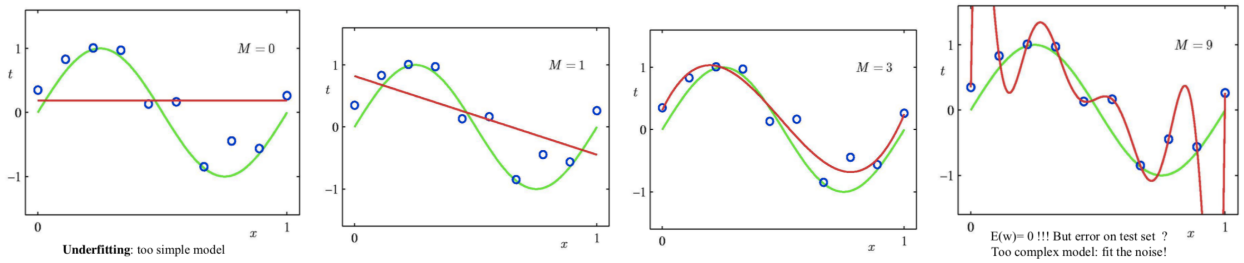
(a) Polynomial Curve Fitting



(b) Sum-of-Squares Error Function

Figure 1.8: A simple parametric model for regression example

- M = 0: in this case we are in **underfitting**, we have a very simple model that cannot fit the data. With M = 0 the model it's just the mean. See figure 1.9a.
- M = 1: Linear model, still a model model, dosen't not fit good the data. See figure 1.9b.
- M = 3, this time, the model seems very good, we have a low error. See figure 1.9c.
- M = 9: Too complex model, it's fit the noise too!, the error on the training test is $E(w) = 0$, but error on the test set? See figure 1.9d. Poor representation of the (green) true function due to **overfitting**.



(a) M = 0 (b) M = 1 (c) M = 3 (d) M = 9

Figure 1.9: The complexity of the hypothesis increase with the degree M

So let's plot Root-Mean-Square(RMS) Error and see how change with the complexity (M) of the model, where $w*$ are the polynomial coefficients:

$$E_{RMS} = \sqrt{2E(w^*)/l}$$

As we can see in figure 1.10 the error is 0 in the training test (overfitting) but the error is really high on the test set. the model is not able to generalize.

Case $l = 15$:
With a 9th order polynomial model ($M = 9$) and more data the model seems to get a better generalize for the future data, but still not good. See figure 1.11a.

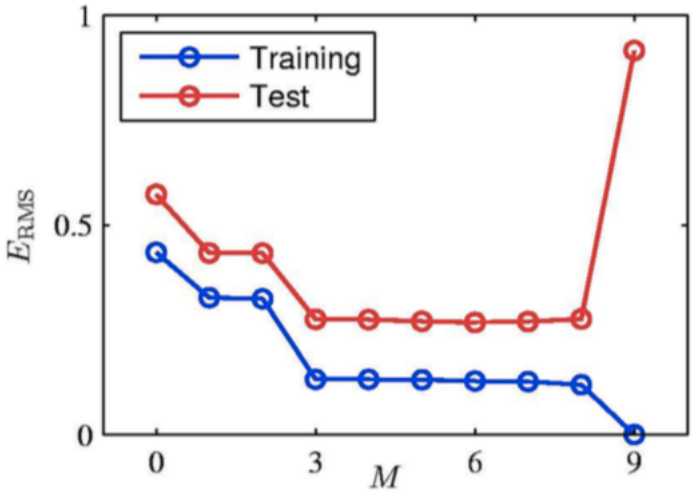
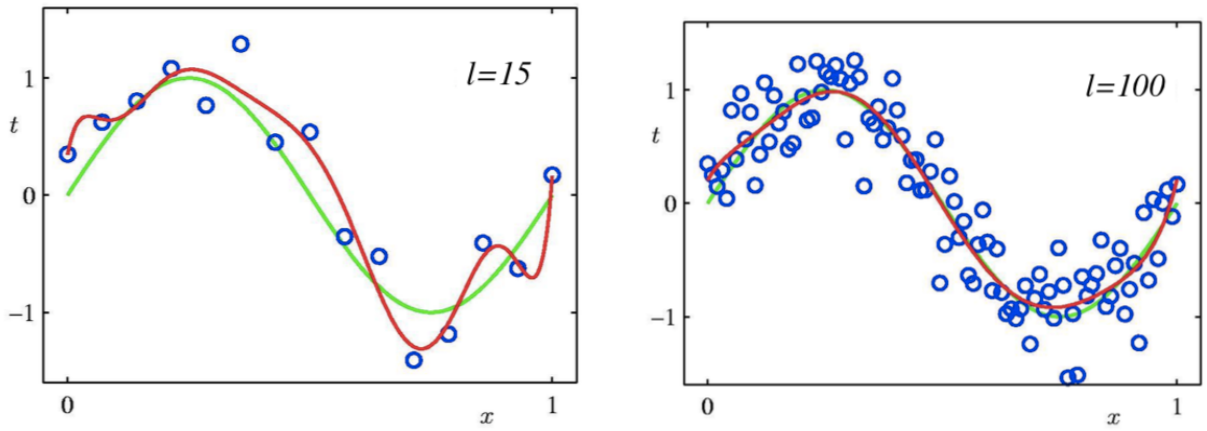


Figure 1.10: Mean Square Error for different values of M on the test set (red) and on training set (blue).

Case $l = 100$:
With a 9th order polynomial model ($M = 9$) and more data the model seems to get a better generalize for the future data, but still not good. See figure 1.11b.



(a) l = 15

(b) l = 100

Figure 1.11: The complexity of the hypothesis with the degree 9 when the number of data change

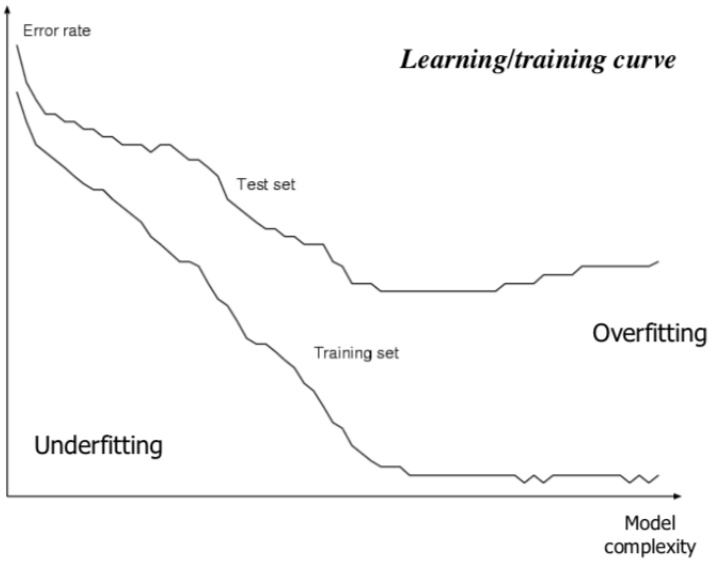
So model complexity and the number of data are very important for the generalization capability of a model, let's put all together:

- The generalization capability (measured as a risk or test error) of a model with respect to the training error and avoid overfitting and underfitting zones
- The role of model complexity
- The role of the number of data

Typical behavior of learning

1.9.2 Toward Statistical Learning Theory (SLT)

Statistical Learning Theory (SLT) is a general theory relating such topics. In a (Simplified) Formal Setting:



1. we want to approximate an unknown function $f(\mathbf{x})$
2. minimize a risk function:
$$R = \int L(d, h(\mathbf{x}))dP(\mathbf{x}, d)$$
This is the **True error** over all the data (that we don't have). d is the value from the teacher and $P(x, d)$ the probability distribution. A loss (or cost) function could be for example: $L(h(\mathbf{x}), d) = (d - h(\mathbf{x}))^2$
3. then search h in H minimizing R

But we have only a finite data set: $TR = (\mathbf{x}_i, d_i)\ \forall i = 1, \ldots, l$. So we can't use R as minimize risk function. To search h, we will minimize the empirical risk (training error), finding the best values for the model free parameters:

$$R_{emp} = \frac{1}{l}\sum_{i=1}^{l}(d_i - h(\mathbf{x}_i))^2.$$

Empirical Risk Minimixzation (ERM) Inductive Principle.

Vapnik-Chervonenkis-dim and SLT: a general theory

we can use R_{emp} to approximate R!

VC-dim: measure complexity of H (flexibility to fit data), e.g. Num. of parameters for polynomials, NN, ...

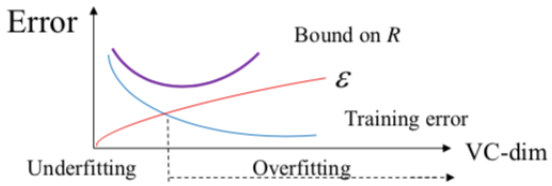
The Vapnik-Chervonenkis bound (VC-bound(s)) is in the form:

$$R \leq R_{emp} + \epsilon(1/l, VC, 1/\delta) \ \ , \ \ \text{with probability } 1 - \delta$$

where $R_{emp} + \epsilon$ is the **guaranteed risk** and ϵ is the **VC-*confidence***

- Higher l (data): lower R
- Higher VC-dim (fix l) : lower R_{emp} but R may increase (overfitting)

Structural Risk Minimization: find a trade-off Concept of control of the model complexity (flexibility): trade-off between model complexity and TR (training) accuracy (fitting).



Definition 1.5 (Model selection)
The model selection consist in choise of the model (H) with the better bound on the true risk.

Example 1.6
It is possible to derive an upper bound of the ideal error which is valid with probability $(1-\delta)$*, delta being arbitrarily small, of the form:*

- General: $R \leq R_{emp} + \varepsilon(1/l, VC, 1/\delta)$
- Example: $R \leq R_{emp} + \varepsilon(VC/l, -\ln(\delta/l))$
- There are different bounds formulations according to different classes of f, of tasks, etc.
- More in general, in other words (simplifying): we can make a good approximation of f from examples, provided we have a good number of data, and the complexity of the model is suitable for the task at hand.

1.9.3 Complexity control

SLT - Statistical Learning Theory:

- It allows formal framing of the problem of generalization and overfitting, providing analytic upper-bound to the risk R for the prediction over all the data, regardless to the type of learning algorithm or details of the model..
- The ML is well founded: the Learning risk can be analytically limited and only few concepts are fundamentals !
- SLT has allowed to leads to new models (e.g SVM) and other methods that directly consider the control of the complexity in the construction of the model
- bases one of the inductive principles on the control of the complexity
- explain the main difference with respect to supporting methods from CM (providing the techniques to perform fitting), apart from modelling aspects

But there are other open questions: what (other) principles are to found the control of the complexity? How to work in practice? How to measure the complexity (or fitting flexibility)? How find the best trade-off between fitting and model complexity?

1.10 VALIDATION

Evaluate generalization capabilities (of your $h(x)$). **Warning**: The performance on training data provide an overoptimistic evaluation

Evaluation of performances for ML systems = Predictive accuracy

Validation phase has two main aims:

- **Model selection**: estimating the performance (generalization error) of different learning models in order to choose the best one (to generalize).
  this includes search the best hyper-parameters of your model (e.g. polynomial order, ...).

  It returns a model

- **Model assessment**: having chosen a final model, estimating/evaluating its prediction error/ risk (generalization error) on new test data (measure of the quality/performance of the ultimately chosen model).

  It returns an estimation

Gold rule: Keep separation between goals and use separate data sets for this two operation.

In an ideal world, for validation we should have:

- a large training set (to find the best hypothesis, see the theory)
- a large validation set for model selection
- a very large external unseen data test set

But with a finite and often small data set we will have just a estimation of the generalization performance. Let's see two basic techniques for validation: Simple hold-out (basic setting) and K-fold Cross Validation.

1.10.1 Hold out cross validation

Hold out: basic setting.
Partition data set D into training set (TR), validation or selection set (VL) and test set (TS). See figure 1.12

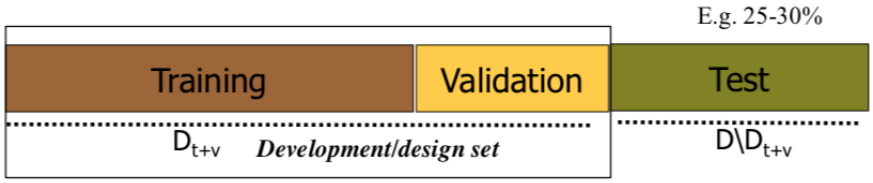


Figure 1.12: Hold out cross validation

- All the three sets are disjoint sets
- Training is used to run the training algorithm
- VL can be used to select the best model (e.g hyper-parameters tuning)
- Test set is not to be used for tuning/selecting the best h: it is only for model assessment

1.10.2 Hold out and K-fold cross validation

When we have a small number of data, Hold out Cross Validation can make insufficient use of data. To solve this problem we can use another technique called: **K-fold Cross-Validation**:

- Split the data set D into k mutually exclusive subsets $D_1, D_2, \ldots, D_k$
- Train the learning algorithm on $\frac{D}{D_i}$ and test it on D_i repeat this phase k time and then we take the mean of the validation results.
- Can be applied for both VL or TS splitting
- It uses all the data for training and validation/testing

But this technique has some issues:

Figure 1.13: k-fold cross validation

- How many folds? 3-fold, 5-fold , 10-fold, ..., 1-leave-out
- Often computationally very expensive (we have to perform training and validation phases k times)

Combinable with validation set, double-K-fold CV, ...

1.10.3 TR/VL/TS by a schema

:

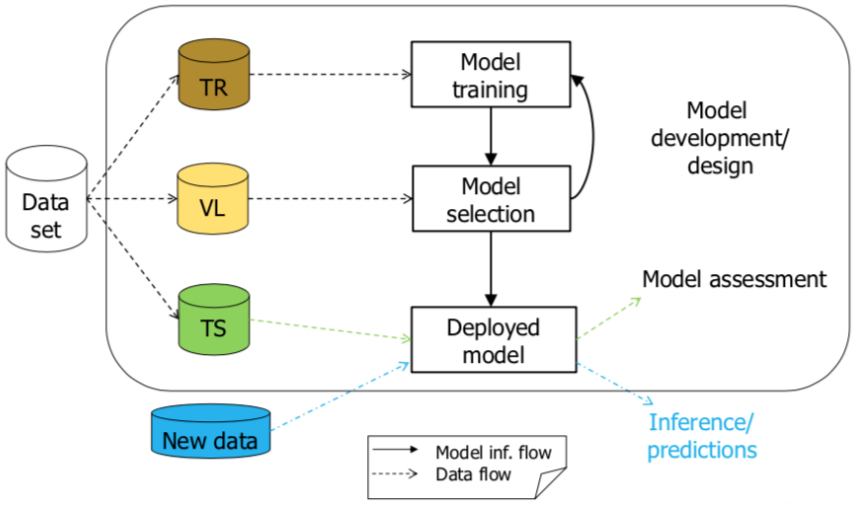


Figure 1.14: TR/VL/TS by a schema

1.10.4 Classification Accuracy

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. See figure 1.15.

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Figure 1.15: Confusion matrix

The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

Definition of the Terms:

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Classification Rate/Accuracy is given by the relation:

$$\frac{TP + TN}{Total = TP + TN + FP + FN}$$

Two importat statistical measures are: *Specificity* and *Sensitivity*.

Specificity

Specificity (also called the true negative rate) measures the proportion of actual negatives that are correctly identified as such (e.g., the percentage of healthy people who are correctly identified as not having the condition).

$$Specificity = \frac{TN}{FP + TN} = 1 - FPR$$

where FPR (fall-out or false positive rate) $= FP/N = FP/(FP + TN)$.

Sensitivity

Sensitivity (also called the true positive rate, the recall, or probability of detection[1] in some fields) measures the proportion of actual positives that are correctly identified as such (e.g., the percentage of sick people who are correctly identified as having the condition).

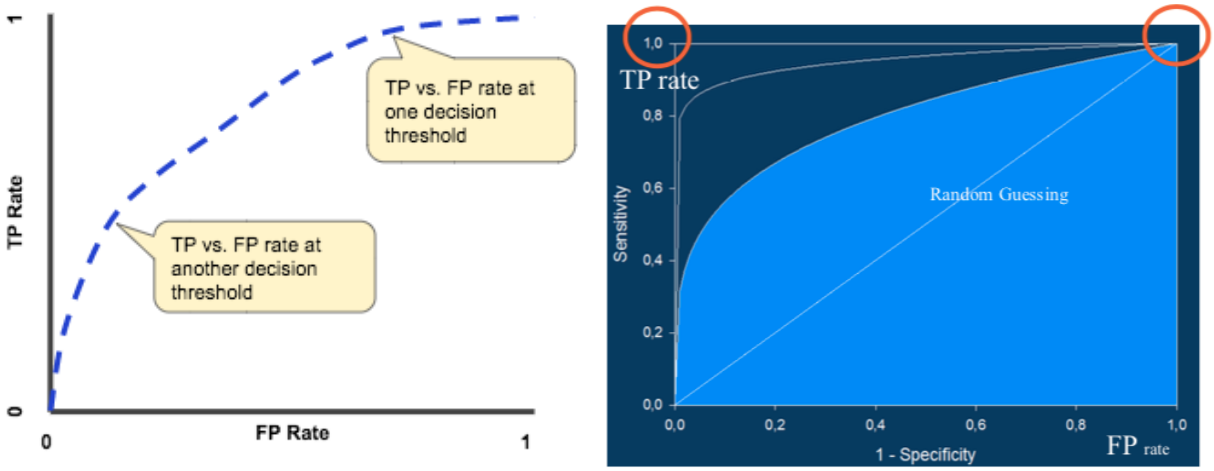
$$Sensitivity = \frac{TP}{TP + FN}$$

NOTE: for binary classif.: 50% correctly classified = "coin" (random guess) predictor. Of course could exists trivial classifier with unbalanced data (e.g. 99% of the data are positive).

ROC Curve

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: Sensitivity (True Positive Rate (TPR)) and False Positive Rate (FPR) (1 - Specificity).

Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. See figure 1.16a.



(a) AUC (Area under the ROC Curve)

(b) AUC

Figure 1.16: ROC curve

AUC stands for "Area under the ROC Curve." and measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). See figure 1.16b.

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

The diagonal corresponds to the worst classifier (random guessing). Better curves have higher AUC (Area Under the Curve). In 1.0 we have the perfect classification.

1.11 The Design Cycle

Data collection: adequately large and representative set of examples for training and test the system

Data representation: Often the most critical phase for an overall success.

- domain dependent, exploit prior knowledge of the application expert
- Feature selection
- Outliers detection
- Other preprocessing: variable scaling, missing data,..

Model choice:

- Statement of the problem.
- Hypothesis formulation: You must know the limits of applicability of your model.
- Complexity control.

Building of the model (core of ML): through the learning algorithm using the training data.

Evaluation: performance = predictive accuracy.

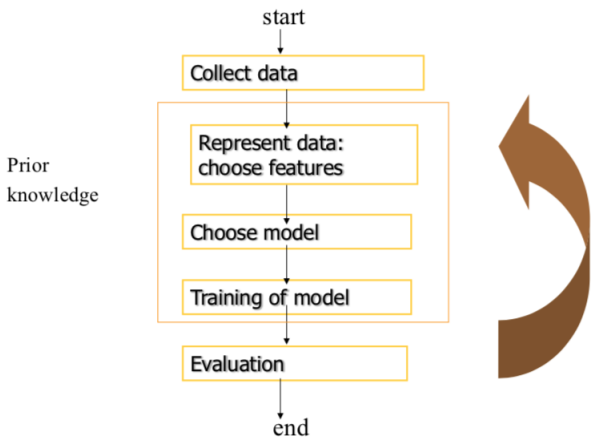


Figure 1.17: Design cycle

1.12 Misinterpretations

For every statistical models (including Data Mining (DM) applications).

Causality is (often) assumed and a set of data representative of the phenomena is needed.

- Not for unrelated variables and for random phenomena (lotteries)
- Uninformative input variables → poor modeling → Poor learning results

Causality cannot be inferred from data analysis alone:

- People in Florida are older(on av.) than in other US states.
- Florida climate causes people to live longer ?

May be there is a statistical dependencies for reasons outside the data.

More specifically for ML:

- Powerful models (even for "garbage" data) → higher risk !
- Not-well validated results: the predicted outcome and the interpretation can be misleading.

2 Linear models

In this section we will going to analyze a linear model for classification and regression (Supervised Learning) both as task in function approximation.

This is a **Parametric model**: A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model.

- **Given**: Training examples as $< input, output >=< x, d >$, we have labeled examples for some for some unknown function f. (For us f is known only at the given example points).
- **Find**: a good approximation to f (that can used for prediction on unseen data $\mathbf{x}'$

The target value **d** (or **t** or **y**), given by the teacher according to $f(\mathbf{x})$, are numerical/ categorical label:

- Classification: $f(x)$ return the (assumed) correct class for x, where $f(x)$ is a discrete valued function.
- Regression: approximate a real-valued target function (in $\mathbb{R}$ o $\mathbb{R}^K$).

Here some Data notation in figure 2.1:

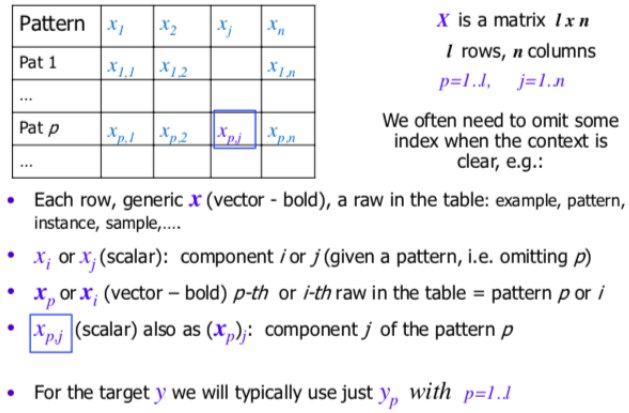


Figure 2.1: Data notation for this section

Now will we start to see linear models for Regression and classification where the H (hypothesis space is linear). Same model can be applied for both tasks and we will see how.

"Despite the great inroads made by modern nonparametric regression techniques, linear models remain important, and so we need to understand them well." (Hastie)

The linear model has been the mainstay of statistics and has a lot of studies and in many books (mathematics, statistics, numerical analysis, applicative fields, ML, ...). This model is also used/included in more complex models.

We will start with the simplest form: linear in the input variables.

2.1 Regression

Let's see first how we can formulate the learning problem as a **Least mean square (LMS)** problem and let's start to see it in a simplified setting: **univariate case**.

x	target
1	2.1
2	3.9
3	6.1
4	8.4
5	9.8
...	...

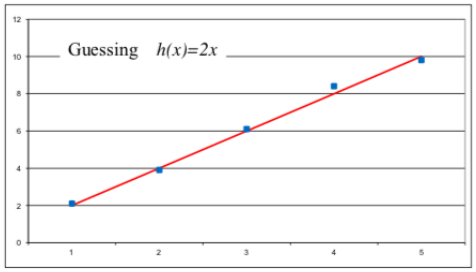


Figure 2.2: Regression Example

A regression task is the process of estimating of a real-value function on the basis of finite set of noisy samples, we have known pairs $(x, f(x) + \text{random noise})$ and we want to find an h (hypothesis) that fit the data. See figure 2.2.

We want to solve it (how to find **w**) in a systematic way.

2.1.1 Univariate Linear Regression

Univariate case, simple linear regression:

- we start with 1 input variable x and 1 output variable y
- we assume a model $h_w(x)$ expressed as:

$$h_w(x) = out = w_1 x + w_0$$

where **w** are real-valued **coefficients/free parameters (weights)**.

The idea is **fitting** the data by a "straight line".

In this case we have Infinite hypothesis space (continuous w values) but we have nice solution from classical math (going back to Gauss/Legendre 1795!). Surprisingly we can "learn" by this basic tool and although simple it include many relevant concept of modern ML and it is a basis of evolved methods in the field.

⇒ **Learning via LMS**:
Learn: means find the **w** such that minimize error/empirical loss (bestdata fitting – on the training set with l examples).

- Given a set of l training example (x_p, y_p)
- find $h_w(x)$ in the form: $w_1 x + w_0$ (hence the values of w) that minimizes the expected loss on the training data.

As Loss function we will use the square of errors, Least (Mean) Square: find w to minimize the residual sum of squares: $argmin_w \|Error(\mathbf{w})\|$

$$Loss(h_{\mathbf{w}}) = E(\mathbf{w}) = \sum_{p=1}^{l} (y_p - h_{\mathbf{w}}(x_p))^2 = \sum_{p=1}^{l} (y_p - (w_1 x_p + w_0))^2$$

Where x_p is p-th input/patter/example, y_p the output for p, w free par., l num. of examples

Note:

- to have the mean, divide by l
- On the notation: Indeed for the univariate case, with 1 variable: $x_p = x_{p,l} = (x_p)_l$

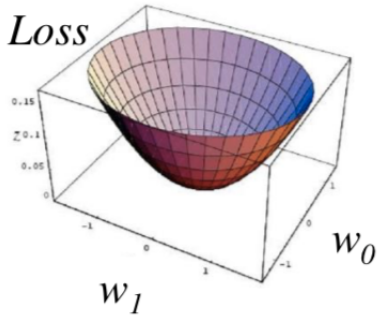
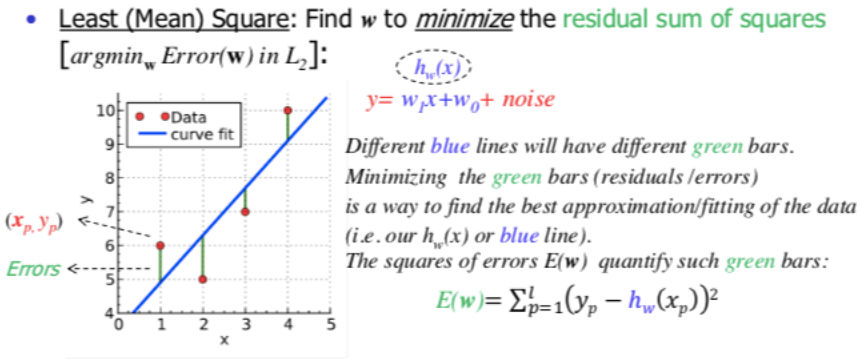


Figure 2.3: Loss function plot with two free parameters

The method of least squares is a standard approach to the approximate solution of over-determined systems, i.e., sets of equations in which there are more equations than unknowns.



⇒ **How to solve?**:
Remember: local minimum as stationary point: the gradient is null:

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = 0 \ , \ i = 1, \ldots, \text{dim\_input} + 1 = 1, \ldots, n + 1$$

For the simple Lin. Regr. (2 free parameters):

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = 0 \ , \ \frac{\partial E(\mathbf{w_0})}{\partial w_1} = 0$$

We have a convex loss function, so there is no local minima and there is a closed form. The solution is:

$$w_1 = \frac{\sum x_p y_p - \frac{1}{l}\sum x_p \sum y_p}{\sum x_p^2 - \frac{1}{l}(\sum x_p)^2} = \frac{\text{Cov}[x,y]}{\text{Var}[x]}, \quad w_0 = \overline{y} - w_1 \overline{x}$$

$$\frac{1}{l}\sum_{p->l} y_p \qquad \frac{1}{l}\sum_{p->l} x_p$$

Let's compute the gradient for 1 (each) patter p:

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial (y - h_{\mathbf{w}}(x))^2}{\partial w_i} = 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - h_{\mathbf{w}}(x))}{\partial w_i} =$$
$$= 2(y - h_{\mathbf{w}(x)})) \frac{\partial (y - (w_1 x + w_0))}{\partial w_i}$$

and we will have:

$$\frac{\partial E(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x)) \ , \frac{\partial E(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x))x$$

then we will sum up for l patterns (x_p, y_p):

$$\frac{\partial E(\mathbf{w})}{\partial w_0} = -2\sum_{p=1}^{l}(y_p - h_{\mathbf{w}}(x_p)) \ , \ \frac{\partial E(\mathbf{w})}{\partial w_1} = -2\sum_{p=1}^{l}(y_p - h_{\mathbf{w}}(x_p))x_p$$

2.1.2 Linear Regression with multidimensional inputs case

Assuming column vector $\mathbf{x}$, $\mathbf{w} \in \mathbb{R}^n$, number of data l:

$$\mathbf{w}^T\mathbf{x} + w_0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Note:

- that often (in NN), as before, the transpose notation T in w^T is omitted
- w_0 is the intercept/threshold/bias/offset (has nothing to do with the inductive bias, bias is just a name here)

Often it is convenient to include the constant $x_0 = 1$ so that we can write:

$$\mathbf{w}^T\mathbf{x} = \mathbf{x}^T\mathbf{w} \ , \ \mathbf{x}^T = [1, x_1, x_2, \ldots, x_n] \text{ and } \mathbf{w^T} = [w_0, w_1, w_2, \ldots, w_n]$$

So, the "linear" model is now:

$$h(\mathbf{x}_p) = \mathbf{x}_p\mathbf{w} = \sum_{i=0}^{n} x_{p,i}w_i \ , \ w_i \text{ continuous (free) parameters "weights"}$$

2.2 Classification

We want now to use the same linear model (with LMS) presented for the regression task and use it for the classification task. The problem in Classification is shown in figure 2.4

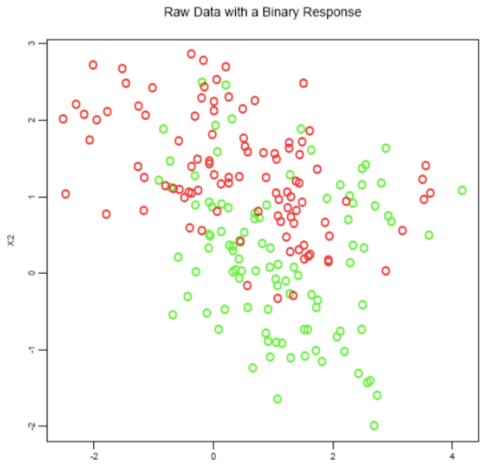


Figure 2.4: A classification problem (Data may be generated by gaussian distribution (for each class) with different means or by a mixture of different low variance gaussian distributions)

We have 200 points generated in $\mathbb{R}^2$ from an unknown distribution, 100 in each of two classes and we want to build a rule to predict the color of future points.

We reuse the linear model for the classification task:

The same models (used for regression) can be used for classification: categorical targets, e.g. $0/1$ or $-1/+1$

- In this case we use an hyperplane ($\mathbf{wx}$) assuming negative or positive values.
- We exploit such models to decide if a point $\mathbf{x}$ belong to positive or negative zone of the hyperplane (to classify it)
- So we want to set $\mathbf{w}$ (by learning) s.t. we get good classification accuracy

Geometrical view: hyperplane:

We define an hyperplaen: $\mathbf{w}^T\mathbf{x}$, see figure 2.5 where our hyperplane is:

$$\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + w_0 = 0$$

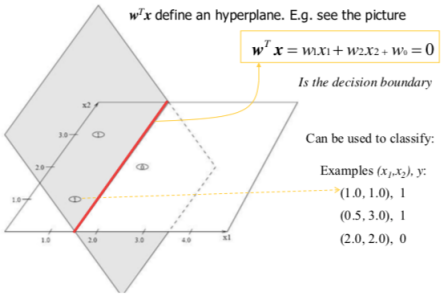


Figure 2.5: Geometrical view: hyperplane

Geometrical view: classifier:
In this case our hypothesis ($h(x)$) is called **Linear threshold unit (LTU)** and is defined as follow:

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{wx} + w_0 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

or

$$h(\boldsymbol{x}) = sign(\boldsymbol{wx} + w_0)$$

Linear threshold unit (LTU)
Indicator functions

In case of w_0 included in $\mathbf{w}$:

$$h(\mathbf{x}_p) = sign(\mathbf{x}_p^T\mathbf{w}) = sign(\sum_{i=0}^{n} x_{p,i}w_i)$$

A Geometrical view of our classifier is shown in figur 2.6

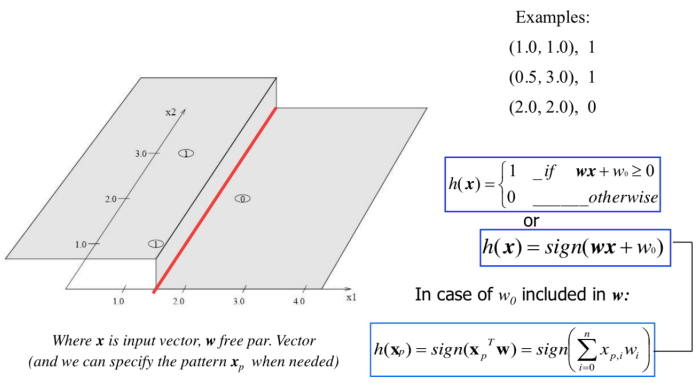
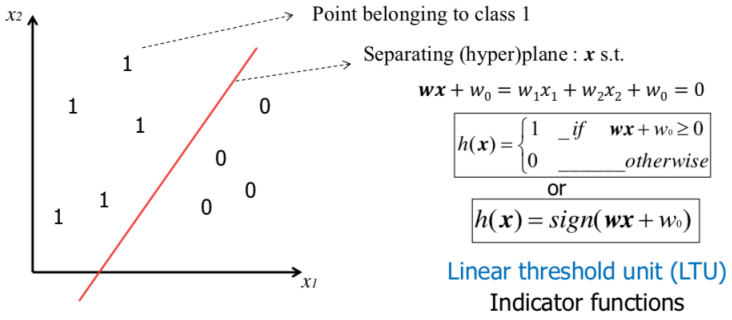


Figure 2.6: Geometrical view: hyperplane

Classification by linear decision boundary:
The classification may be viewed as the allocation of the input space in decision regions (e.g. 0/1) and the linear decision boundary can solve a linearly separable problem.

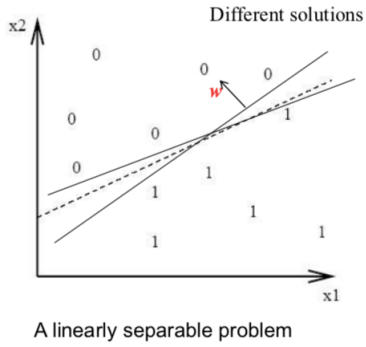
Example: linear separator on 2-dim instance space where our hypothesis space is composed by all the possible hyperplane (how the weight (**w**) change).

$$\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2 \ , \ f(x) = 0/1 \text{ (or -1/+1)}$$



In general the solution is not unique: there are many possible hyperplanes separating, let's see some hyperplane properties and how the hyperplane change with different **w** values.

Hyperplane properties:

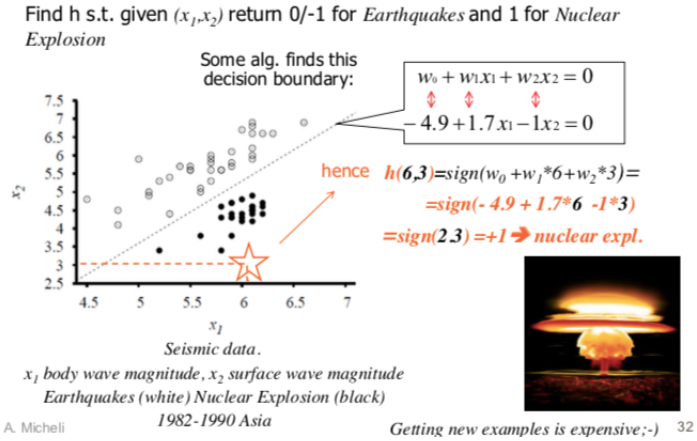


- If $w_0 = 0$ the line goes through the origin of the coordinate system.
- If $n > 2 \quad \rightarrow \quad$ hyperplane
- Scaling freedom: the same hyperplane multiplying $\mathbf{w}$ by K
- $\mathbf{w}$ is a vector orthogonal to the hyperplane, given $\mathbf{x_a}, \mathbf{x_b}$ (belonging to the hyperplane):

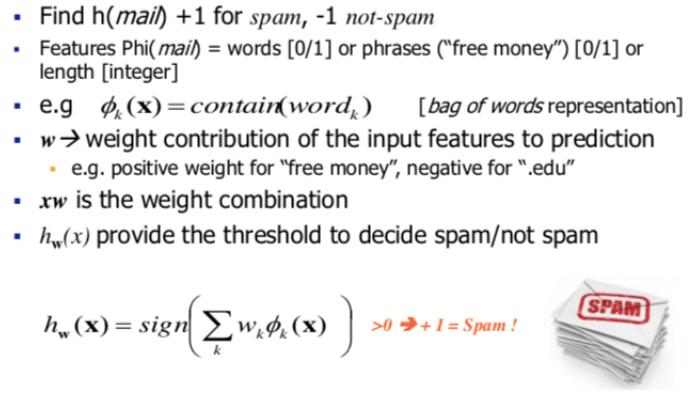
$$\mathbf{x_a^T w} + w_0 = 0 \; ; \; \mathbf{x_a^T w} + w_0 = 0 \rightarrow \mathbf{w^T}(\mathbf{x_a} - \mathbf{x_b}) = 0 \rightarrow \text{orthogonal vectors}$$

Here two examples:

(AIMA) Classify a new data (x_1, x_2)



Spam



2.3 Learning Algorithms

A learning algorithm is divided into two types:

- **Eager**: Analyze the training data and construct an explicit hypothesis.
- **Lazy**: Store the training data and wait until a test data point is presented, then construct an ad hoc hypothesis to classify that one data point.

The linear model expected to construct an explict hypothesis from the training set, so the learning algorithm for the linear model is **Eager**.

We are going to introduce 2 *learning algorithms* for the regression and for the classification task using a linear model, both based on *LMS*.

- A direct approach based on **normal equation** solution
- An iterative approach based on **gradient descent**

The learning algorithm has

We start redefining the learning problem and the loss for them (for l data and multidimensional inputs).

The learning problem (classification tasks)
Given a set of l training example $(\mathbf{x}_i, y_i)$ and a loss function (measure) L, we want to **find** the weight vector $\mathbf{w}$ that minimizes the expected loss on the training data.

$$R_{emp} = 1/l \sum_{i=1}^{l} L(h(\mathbf{x}_i), y_i)$$

For classification: Using a piecewise constant (over $sign(\mathbf{w^T x})$) for the loss can make this a difficult problem because it is not continuous and differentiable. Assume we still use the least squares (as for the regression case).

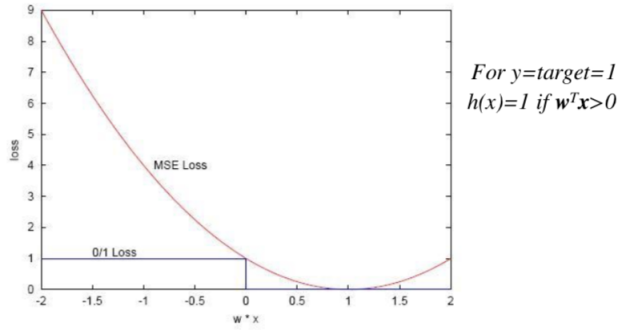


Figure 2.7: Both satisfies the minimization of error

Initially, we can make the optimization problem easier by replacing the original objective function **L** by a smooth, differentiable function. For example, consider the mean squared error. See figure 2.7.

Learning (a classifier) by Least Squares:

Given a set of l training example $(\mathbf{x}_i, y_i)$, **Find** optimal values for w (for fitting of Training data) by using the **least squares** that minimize the residual sum of squares:

$$E(\mathbf{w}) = 1/l \sum_{i=1}^{l} (y_i - \mathbf{x}_i^T \mathbf{w})^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

Min error: if $y_i = 1$ then $x_i^T w \to 1$; if $y_i = 0$ then $x_i^T w \to 0$

In $E(\mathbf{W})$ we do not use $h(\mathbf{x})$, as for regression, but to hold a continuous differentiable loss (because $h(\mathbf{x}) = sign(\mathbf{wx})$ for classification) we will use $\mathbf{x}_i^T \mathbf{w}$
This is a *quadratic function* so the minimum always exists (but may be not unique).
(X is a matrix $\mathbf{l} \mathbf{x} \mathbf{n}$ with a row for each input vector $\mathbf{x}_i$).

2.3.1 Normal equation and direct approach solution

Differentiating $E(\mathbf{w})$ with respect to $\mathbf{w}$ we get the normal equation (point with gradient of E w.r.t $\mathbf{w} = 0$):

$$(\mathbf{X^T X})\mathbf{w} = \mathbf{X^T y}$$

Proof. Now we want to compute $\frac{\partial E(\mathbf{w})}{\mathbf{w}_j} = 0 \ , \ \ j = 0, \ldots, n$

$E(\mathbf{w}) = \sum_{i=1}^{l} (y_i - \sum_{t=0}^{n} w_t x_{i,t})^2 = \sum_{i=1}^{l} (\delta_i(\mathbf{w}))^2$ (l patterns)

$$\frac{\partial E(\mathbf{w})}{\mathbf{w}_j} = 2 \sum_{i=1}^{l} \delta_i(\mathbf{w}) \frac{\partial(\delta_i(\mathbf{w}))}{\partial \mathbf{w}_j} = 2 \sum_{i=1}^{l} \delta_i(\mathbf{w}) \frac{\partial(y_i - \sum_{t=0}^{n} w_t x_{i,t})}{\partial \mathbf{w}_j} =$$

$$= 2 \sum_{i=1}^{l} \delta_i(\mathbf{w})(-x_{i,j}) = -2 \sum_{i=1}^{l} x_{i,j} \delta_i(\mathbf{w}) = -2 \sum_{i=1}^{l} x_{i,j} (y_i - \sum_{t=0}^{n} w_t x_{i,t}) = 0$$

$$\sum_{i=1}^{l} x_{i,j} y_i = \sum_{i=1}^{l} \sum_{t=1}^{n} w_t x_{i,t} x_{i,j} \ \ , \ \ j = 0, \ldots, n$$

$$\mathbf{X^T y} = (\mathbf{X^T X})\mathbf{w}$$

□

So, if $(\mathbf{X^T X})$ is not singular the unique solution is given by:

$$\mathbf{w} = (\mathbf{X^T X})^{-1} \mathbf{X^T y} = \mathbf{X}^+ \mathbf{y}$$

Else the solution are infinite (satisfying the normal equation):

we can choose the min norm (w) solution.

note: ($\mathbf{X}^+$ is the Moore-Penrose pseudoinverse also if X is not invertible)

Direct approach by SVD:
The Singular Value Decomposition can be used for computing the pseudoinverse of a matrix:

$$\boldsymbol{X} = \boldsymbol{U}\underset{\text{diagonal}}{\boldsymbol{\Sigma}}\boldsymbol{V}^T \Rightarrow \boldsymbol{X}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T$$
by replacing every nonzero entry by its reciprocal

Moreover we can apply directly SVD to compute $\mathbf{w} = \mathbf{X}^+ \mathbf{y}$ obtaining the minimal norm (on w) solution of least squares problem.

2.3.2 Gradient descent

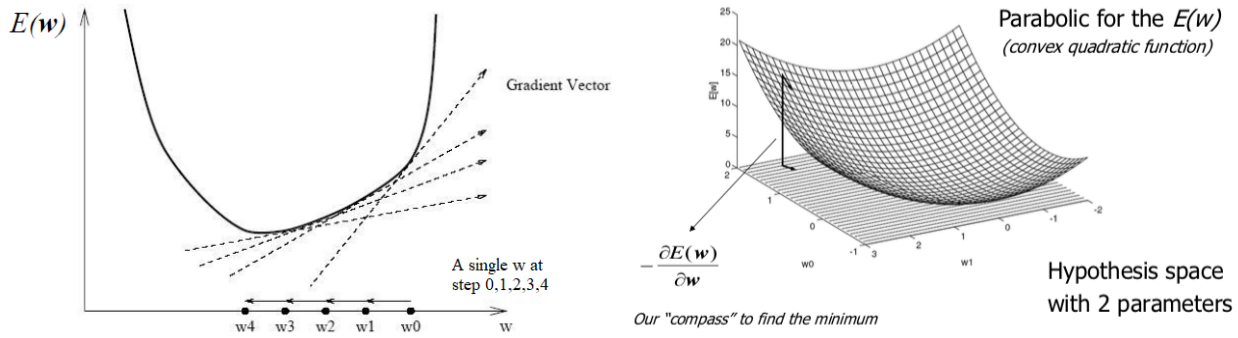
An iterative approach based on *gradient descent*.
Previous derivation suggest the line to construct an iterative algorithm based on :

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_j} = -2\sum_{i=1}^{l}(y_i - \mathbf{x}_i^T\mathbf{w})(\mathbf{x}_i)_j$$

Where

- $\mathbf{x}_i$: i-th input pattern
- y_i: the output for p
- $\mathbf{w}$: free parameters
- l: numbers of examples
- $(\mathbf{x}_i)_j$: component j of pattern i

Gradient (ascent direction): we can move toward the minimum with a gradient descent (- gradient of $E(\mathbf{w})$).
Local search: begins with initial weight vector. Modifies it iteratively to decrease up to minimize the error function (steepest descent).



(a) Gradient descent

(b) H for 2 –w linear model

Figure 2.8: Gradient

Delta Rule
A training rule whose key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
The ≪movements≫ will be made iteratively according to:

$$\mathbf{w}_{new} = \mathbf{w}_{old} + eta \times \Delta\mathbf{w}$$

where eta is the step size, $0 < eta < 1$ or a value

learning rate (η) - eta =: speed/stability trade-off and is the step size: can be (gradually) decreased to zero (guarantee convergence, avoiding oscillation around the min.): many variants will be introduced later.

In figure 2.9 is shown how the gradient descent change with different learning rate and figure 2.11 is shown how the learning curve can change.

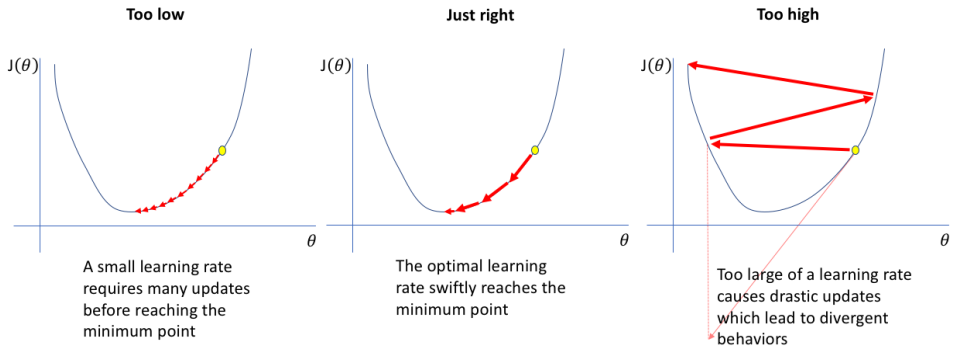


Figure 2.9: Learning rate

Gradient descent algorithm

1) Start with weight vector $w_{initial}$ (small), fix *eta (0<eta<1)*.
2) Compute $\Delta w = -$"gradient of $E(w)$" $= -\frac{\partial E(w)}{\partial w}$ (for each w_j)
3) Compute $w_{new} = w_{old} + eta * \Delta w$ (for each w_j) → Learning rule
   where *eta* is the "step size" parameter
4) Repeat (2) until convergence or $E(w)$ is "sufficiently small"

In the stardard case we use $\Delta w/l$: least mean squares (when you divide by l). It's good to normalize the value of the graident, otherwise could be too large and could lead to oscillations (bad learning curve)

There are two versions of the algorithm:

- For **batch version** the gradient is the sum over all the l patterns (using p):

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}_i} = -2\sum_{p=1}^{l}(y_p - \mathbf{x}_p^T\mathbf{w})x_{p,i}$$

  So, to calculate the gradient of the cost function, we need to sum the cost of each pattern. If we have 3 million patterns, we have to loop through 3 million times or use the dot product.
  Note that:
  - $x_{p,i}$ is the component i of pattern p
  - 2 is constant that can be ignored to develop the algorithm
  - we typically use *LMS* (use $1/l$ in front of the sum)
  - provide a more "precise" evaluation of the gradient over a set of l data
  - We upgrade the weights after this sum
- For the **on-line/stochastic version** we upgrade the weights with the error that is computed for each pattern.

$$\frac{\partial E_p(\mathbf{w})}{\partial \mathbf{w}_i} = -2(y_p - \mathbf{x}_p^T\mathbf{w})x_{p,i}$$

  In SGD weights are updated upon examining each training example so, we only use 1 example for each learning step. Because it's using only one example at a time, its path to the minima more random

than that of the batch gradient. Usually, before for-looping, you need to randomly shuffle the training examples.

- We will see intermediate cases later (as **mini-batch**): Mini-batch gradient descent uses n data points (instead of 1 sample in SGD) at each iteration.

Here an example showing how the trajectory change with different gradient descent algorithm:

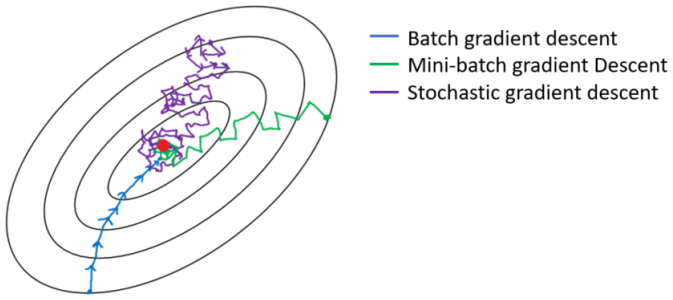


Figure 2.10: Gradient descent variants trajectory towards minimum

Learning curve examples: the curve can change with different gradient algorithm and different value of the learning rate:

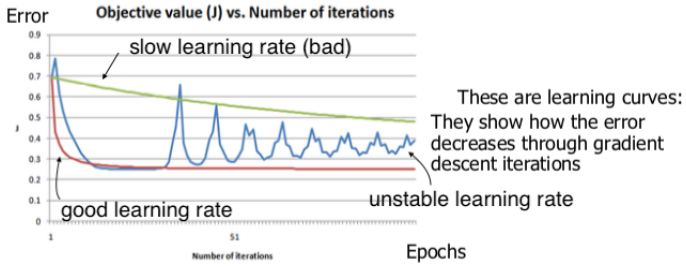


Figure 2.11: Learning curve examples

Gradient descent as Error correction rule

$$\Delta w_j = \sum_{p=1}^{l} (y_p - \mathbf{x}_p^T \mathbf{w})(\mathbf{x}_p)_j$$

This is an "error correction" rule (Widrow-Hoff) that change the w proportionally to the error (target-output):
For example:

- $(target - output) = err = 0 \rightarrow$ no correction
- $(input_j > 0)\ if\ err\ +$ (output is too low) :

$$increase\ w_j \rightarrow\ increase\ output \rightarrow less\ err$$

- $(input_j > 0)\ if\ err\ -$ (output is too high) :

$$decrease\ w_j \rightarrow\ reduce\ output \rightarrow less\ err$$

- $(input_j < 0)\ if\ err\ +$ (output is too low) :

$$decrease\ w_j \rightarrow\ increase\ output \rightarrow less\ err$$

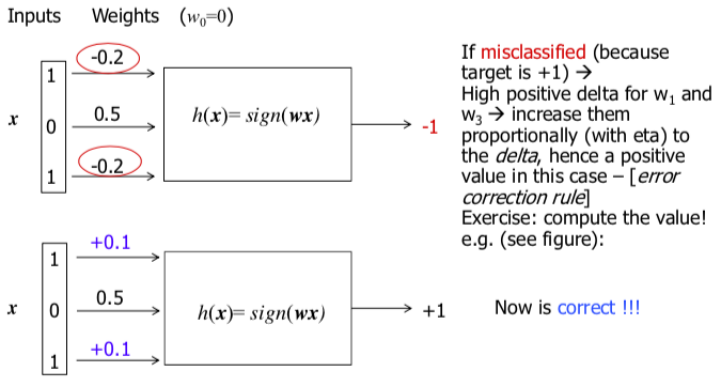
- $(input_j < 0)\ if\ err\ -$ (output is too high) :

$$increase\ w_j \rightarrow\ reduce\ output \rightarrow less\ err$$

It allow us to search through an *infinite hypothesis* space and it can be easily always applied for *continues* H and differentiable loss.

NOTE: Is the gradient descent efficient? Many improvement are possible: Newton and quasi-newton methods, Conjugate Gradient, ...

Here and example of Delta-W as Error Correction Learning rule:



Summarizing

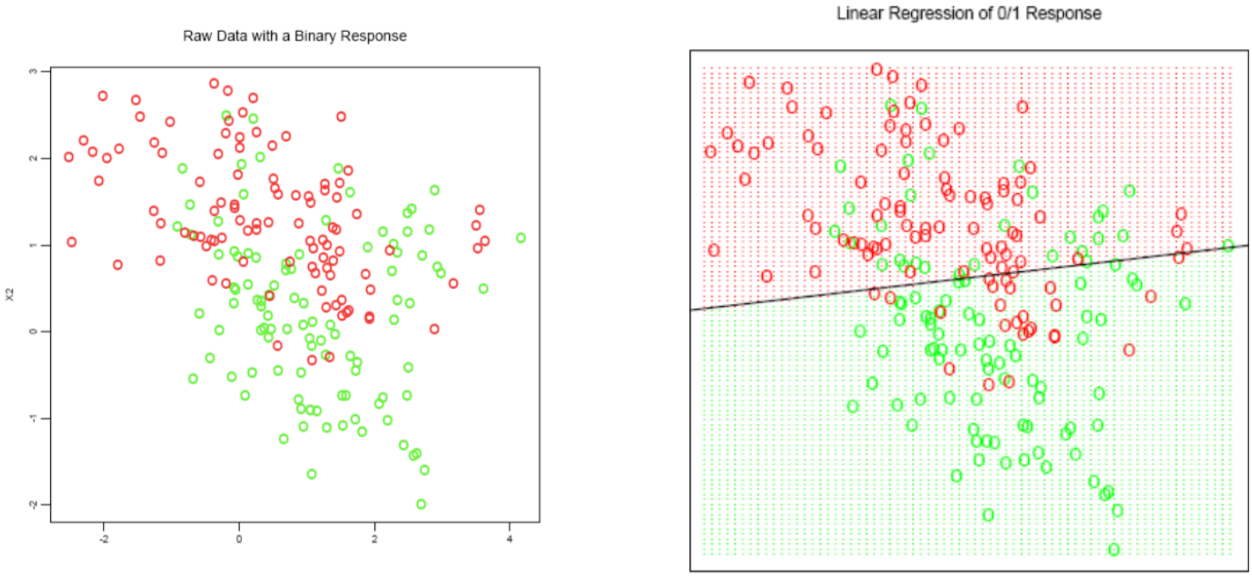
- Model trained (on tr set) with LS (LMS) on **wx** by the simple gradient descent algorithm used for linear regression
- Model used for classification applying the threshold function $h(\mathbf{x}) = sign(\mathbf{wx})$
- The error can be computed as classification error or number of misclassified patterns (not only by the Mean Square Error)

$$L(h(\mathbf{x}_p), d_p) = \begin{cases} 0 & \_if \quad h(\mathbf{x}_p) = d_p \\ 1 & \_\_\_\_\_otherwise \end{cases} \qquad mean\_err = \frac{1}{l} \underbrace{\sum_{i=1}^{l} L(h(\mathbf{x}_i), d_i)}_{num\_err}$$

- ACCURACY = mean of correctly classified = $(l - num\_err)/l$

2.4 Linear model on a classification problem

Let's analyze the problem shown in figure 2.12a problem and let's apply the linear model on the training data set.



(a) Raw data

(b) After LMS algorithm

Figure 2.12: Solution of a classification example with LMS

A classification example in two dimensions. The classes are coded as a binary variable. In figure 2.12b we can see the result of the linear model: The line is the decision boundary defined by $\mathbf{x^T w} = 0.5$.

$$h(\mathbf{x}) = \begin{cases} 1 & if \ \ \mathbf{x^T w} > 0.5 \\ 0 & otherwise \end{cases}$$

The decision boundary is $\{\mathbf{x} \mid \mathbf{x^T w} = 0.5\}$ is linear (and seems to make many errors on the training data). Is it true?

Good or bad approximation: Possible scenarios (we know the true target function!):

- Scenario 1: The data in each class are generated from a Gaussian distribution with uncorrelated components, same variances, and different means.

  In this case $\Rightarrow$ the linear regression rule (by LS) is almost optimal (is the best one can do). The region of overlap is inevitable (due to errors in the input data).

- Scenario 2: The data in each class are generated from a mixture of 10 gaussians in each class.

  In this case $\Rightarrow$ the linear model is far too rigid: next models for it!

2.5 Linear regression (in statistics)

In statistics, **linear regression** is used for two things:

- to construct a simple formula that will predict a value or values for a variable given the value of another variable.
- to test whether and how a given variable is related to another variable or variables.

Statistical Parametric models: note that, in general,"**linear**" does not refer to this straight line, but rather to the way in which the regression coefficients occur in the regression equation. (See next for "linear basis expansion")

NOTE: Least squares corresponds to the maximum likelihood criterion if the experimental errors have a normal distribution.

2.6 Linear model (in ML): Inductive Bias (alla Mitchell)

In the Linear model we have two Bias:

- **Language bias**: the H is a set of linear functions (may be very restrictive and rigid)
- **Search bias**: ordered search guided by the Least Squares minimization goal: for instance we could prefer a different method to obtain a restriction on the values of parameters, achieving a different solutions with other properties...

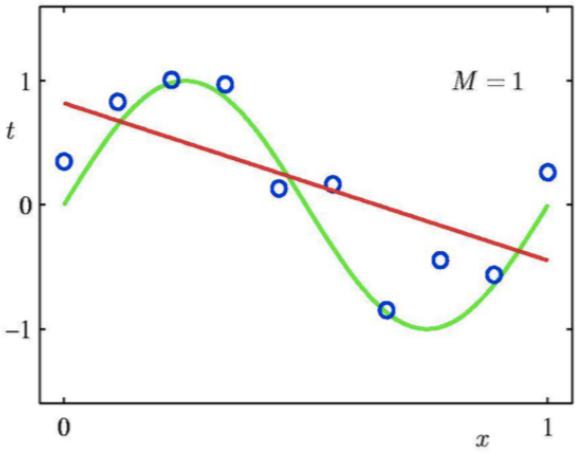
It show that even for a "simple" model there are many possibilities. We still need a principled approach! (see theory of ML)...

2.7 Limitations

This linear model has some limitation: It's a **rigid model**.

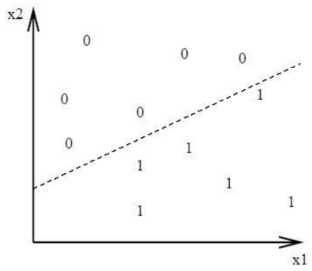
■ **Regression tasks for non linear problems**

In this case we have a very poor solution, a linear model can't fit a non linear problem! we have to find a solution for it!

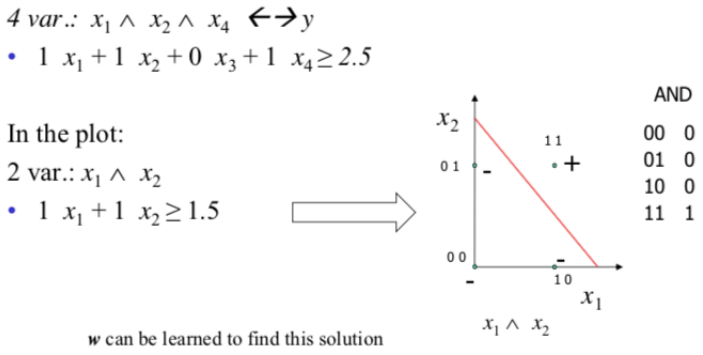


■Classification tasks

- In geometry, two set of points in a two-dimensional plot are **linearly separable** when the two sets of points can be completely separated by a single line.
- In general, two groups are linearly separable in $n-dimensional$ space if they can be separated by an $(n-1)-dimensional$ hyperplane.
- The linear decision boundary can provide exact solutions only for linearly separable sets of points.

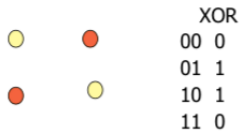


Example: Conjunctions
We can represent conjunctions by the linear models:



Example: 3 point
Given 3 points, we can find a separation plane only if they are not aligned, otherwise no (3 aligned points with 0 in the middle and others 1):

Example: 4 points
Given 4 points we can't always find a separtion plane (we can find a labeling such that the linear classifier fails to be perfect), e.g xor:



2.8 A generalization

The linear model is a very rigid model, here we will show a basis expansion. This allow to get more **Flexibility**.

■Regression task
Basis transformation(**linear basis expansion**) :

$$h_w(\mathbf{x}) = \sum_{k=0}^{K} w_k \phi_k(\mathbf{x})$$

Augment the input vector with additional variables ($K > n$) which are transformations of $\mathbf{x}$ according to a function $\phi_k : R^n \to R$. ϕ can be every function.

E.g:

- Polynomial representation of x: $\phi(x) = x_j^2$ or $\phi(x) = x_j x_i$, or other functions
- Non-linear transformation of single inputs: $\phi(x) = log(x_j)$, $\phi(x) = root(x_j)$, ...
- Non-linear transformation of multiple input: $\phi(x) = \|x\|$
- Splines, ...

This generalization is called: **dictionary** approaches and the model is linear in the parameters (also in phi, not in x!), so we can use the same learning algorithm as before!

Example:

- 1-dim x : $\phi_j(x) = x^j$ ($1 - dim$ polynomial regression ($K = M$)):

$$h(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- any other, $\phi(\mathbf{x}) = \phi([x_1, x_2, x_3])$:

$$h(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 log(x_2) + w_4 log(x_3) + w_5 (x_2 x_3) + w_0$$

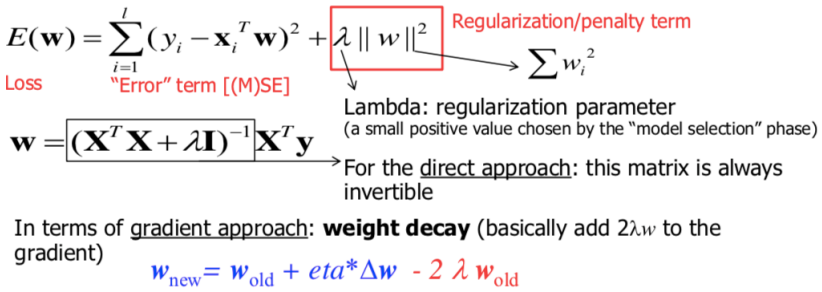
For *classification task* we add sign before the sum.

- **PROS**: Can model more complicated relationships (than linear). It is more expressive with more flexibility.
- **CONS**: With *large* basis of functions, we easily risk *overfitting*, hence we require methods for *controlling the complexity* of the model. Furthermore:
  - Phi are fixed before observing training data, it's hard to decide which ϕ will be good for our data! (versus adaptive /non-linear in parameters e.g. in Neural Network)
  - *Curse of dimensionality* (the volume of the problem space increases so fast that the available data become sparse).

2.9 Improvements

2.9.1 How to control model complexity? Regularization

■Ridge regression (Tikhonov regularization)
smoothed model → e.g. lower variance



Lambda (λ) regularization parameter:

$$0 \leq \lambda < 1$$

Possible to add constraints to the sum of value of $|w_j|$ favouring "sparse" models e.g. with less terms due to weights $w_j = 0$ (i.e. less complex solutions): ridge regression [or lasso, et al. (by different norms)]

- The penalty term *penalizes high value of the weights* and tends to drive all the weights to smaller values (some weights values can go even to zero).
- It *implements* a control of the model complexity
- This leads to a model with *less VC-Dim.*
- λ values can rule the underfitting/overfitting cases.

Tikhonov regularization can be applied for every function and allow us to control the VC-dim. It's better to have a big H space and then control the search bias with λ.

■**Example of control of model complexity:**
Let's suppose to have a regression task and we will use this $h(\mathbf{x})$:

$$h(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$x : \phi_j(x) = x^j (1 - dim \text{ polynomial regression } (M = 9))$$

Let's see how we can control the complexity of the model:

- in figure 2.13a we can see the case where $\lambda = 0$ and there is no regularization. In this case how $h(\mathbf{x})$ is in overfitting and the error $E(w) = 0$ on the training set. This model is too complex and is able to fit the noise too. (we have to control the complexity of the model in this case
- in figure 2.13b we use a small positive lambda ($\lambda = 0.0000000152$. The model has been regularized it and the VC-dim has been reduced too. This model seems to works much better.
- in figure 2.13c we use a very high lambda, close to 1 ($ln\lambda = 0$) and the model goes in underfitting.

From the three previous cases we understand that we need a trade-off and we need to find the right lamda value that does not make us go underfitting or overfitting.

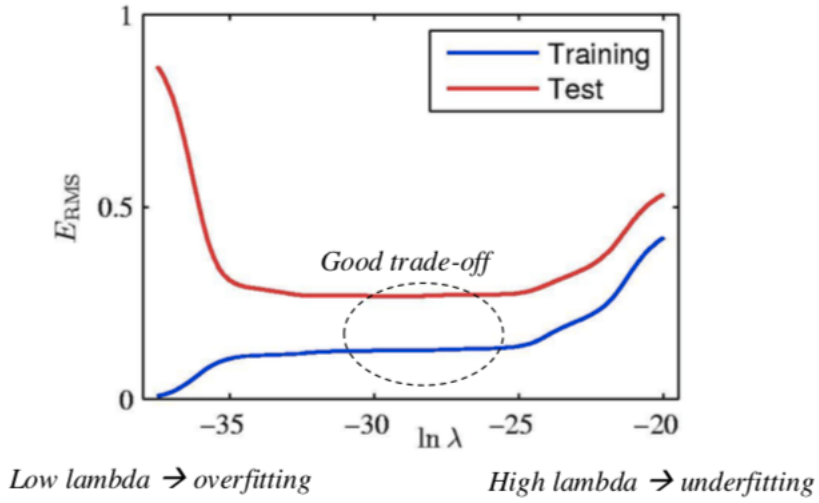
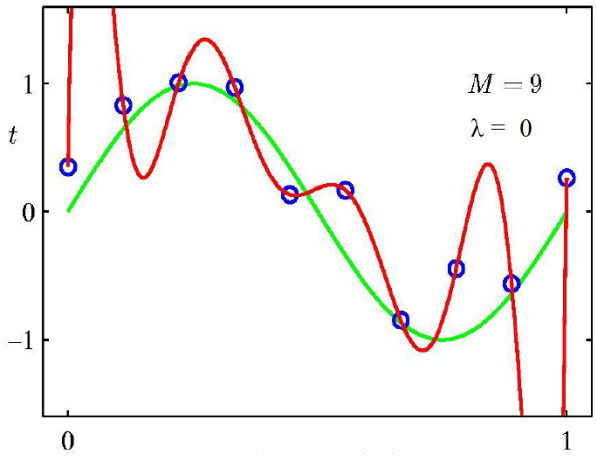
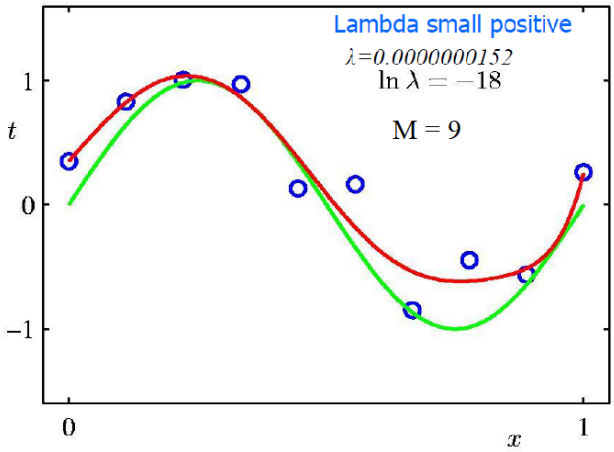


Figure 2.14: Regularization: E_{RMS} vs $ln\lambda$

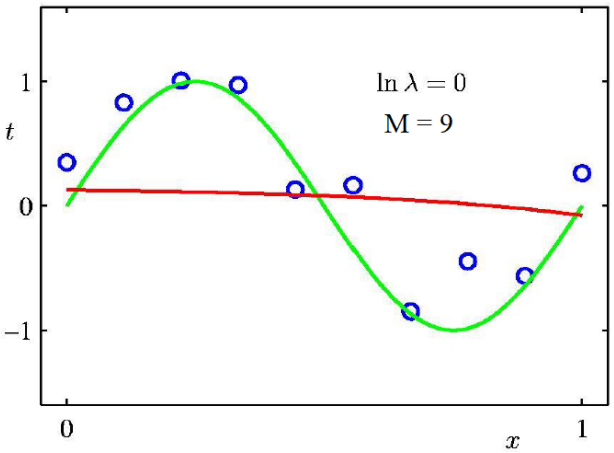
In figure 2.14 we can see how the E_{RMS} change with different value of λ and figure 2.15 shows how the value of weights change with different value of λ.



(a) Lambda = 0



(b) $ln\lambda = -18$: lambda small positive



(c) $ln\lambda = 0$: very high lambda, close to 1

Figure 2.13: 9th Order Polynomial with control of complexity

0 lambda ↖ ↗ Very high lambda

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Figure 2.15: Polynomial Coefficients

2.9.2 Other Regularization Technology for Linear Models

There is other technology used for regularization that we will introduce in later sections:

- Ridge regression ($\|\cdot\|_2$)
- Lasso ($\|\cdot\|_1$)
- Elastic nets (use both $\|\cdot\|_1$ and $\|\cdot\|_2$)

The L2 norm penalizes the square value of the weight and tends to drive all the weights to smaller values. On the other hand, the L1 norm penalizes the absolute value of the weight and tends to drive some weights to exactly zero (while allowing some weights to be large) → toward feature selection!

NOTE: Unfortunately $\|\cdot\|_1$ (absolute value) introduce a non differentiable so the loss needs other approaches.

2.9.3 Others Improvements

- Inputs with added noise (data augmentation)
- Derived inputs: a small number of new variables is used in place of the x inputs, which are a linear combination of x.
  - *Principal Component Regression*
  - *Partial Least Squares*

2.10 Multi-class task

There is two very simple approaches for multi-class:

- **OVA (one-vs-all)**: a discriminant function for each class, built on top of real-valued binary classifiers:
  - train K different binary classifiers, each one trained to distinguish the examples in a single class from the examples in all remaining classes.
  - to classify a new example, the K classifiers are run, and the classifier which outputs the largest (most positive) value is chosen.

$$h(\mathbf{x}) = \arg\max_i h_i(\mathbf{x})$$

  e.g. Class 1-of-K rep: red, green, blue → (0,0,1), (0,1,0), (1,0,0). → solve 3 linear models.

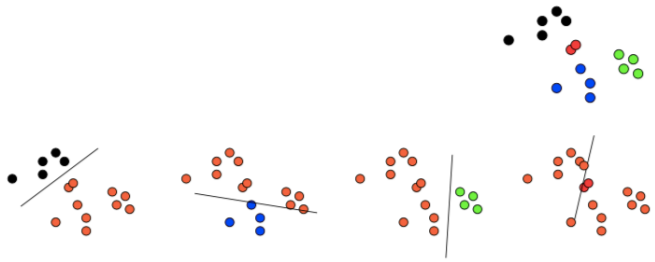
- **AVA (all-vs-all = one-versus-one)**: Each classifier separates a pair of classes. Build $K(K-1)$ classifiers K-way multiclass problem, one classifier to distinguish each pair of classes i and j.
  - to classify a new example, all the classifiers are run, and the winner is the one with the max sum of outputs versus all the other classes OR the class with most votes

$$h(\mathbf{x}) = \arg\ \max_i \left(\sum_j h_{ij}(\mathbf{x})\right)$$

  - training data set for each classifier is much smaller

OVA, AVA suffers from ambiguities in that some regions of its input space may receive the same number of votes.

Criticism: The problem can become not easy



- Masking: classes can be masked by others (for high K)
- A wide array of more sophisticated approaches for multiclass classification exists ...
- Some models can deal directly with multi-output

2.11 Other learner models for classification

- Linear Discriminant Analysis (also multi-class)
- Logistic regression
  $P(y|\mathbf{x})$ starting from modeling the class density as a know density

Extensions: ML Models
(pro future) (#)

Dip. Informatica
University of Pisa

In ML course:

- Perceptron (Rosenblatt 1958, biological inspiration):
  - "minimize (only) misclassifications" algorithm
  - basis for Neural Networks (set of units with layers)
  - Adaptive *basis expansions (phi learned by training)*
    - Feature representation learning in each layer (deep learning concept)
  - Gradient descent approach for learning
- SVM (Vapnik 1996):
  - *regularization* via the concept of maximum margin:Maximize the gap (margin) between the two classes on the training data.
  - enlarge the feature space via *basis expansions* (e.g. polynomials).
- NN and SVM models realize (also) a flexible ***non-linear*** approximation for classification and regression problems

ML Course structure
Discussion on Linear Model

In the *file rouge* of ML, 2 main concepts up to now:

- Basis expansion → (implement) more flexibility
- Regularization → (implement) control of complexity

- The cases of the *linear models* provided an *instance* routed in the classical mathematical approaches but the we will find again these concepts, by different models and implemented in different/similar forms, in the modern ML (e.g. for NN & SVM in our course)!

- Now we move on the other extreme toward a very flexible approach